EFFICIENT ALGORITHMS FOR ASSORTATIVE EDGE SWITCH IN LARGE LABELED NETWORKS

Hasanuzzaman Bhuiyan
Department of Computer Science
Network Dynamics and Simulation Science Laboratory
Biocomplexity Institute of Virginia Tech
Blacksburg, VA, USA
mhb@bi.vt.edu

Maleq Khan
Department of Electrical Engineering
and Computer Science
Texas A&M University—Kingsville
Kingsville, TX, USA
maleq.khan@tamuk.edu

Madhav Marathe
Department of Computer Science
Network Dynamics and Simulation Science Laboratory
Biocomplexity Institute of Virginia Tech
Blacksburg, VA, USA
mmarathe@bi.vt.edu

ABSTRACT

An assortative edge switch is an operation on a labeled network, where two edges are randomly selected and the end vertices are swapped with each other if the labels of the end vertices of the edges remain invariant. Assortative edge switch has important applications in studying the mixing pattern and dynamic behavior of social networks, modeling and analyzing dynamic networks, and generating random networks. In this paper, we present an efficient sequential algorithm and a distributed-memory parallel algorithm for assortative edge switch. To our knowledge, they are the first efficient algorithms for this problem. The dependencies among successive assortative edge switch operations, the requirement of maintaining the assortative coefficient invariant, keeping the network simple, and balancing the computation loads among the processors pose significant challenges in designing a parallel algorithm. Our parallel algorithm achieves a speedup of 68 – 772 with 1024 processors for a wide variety of networks.

Keywords: assortative edge switch, random network generation, network dynamics, parallel algorithms.

1 INTRODUCTION

Networks (or graphs) are simple representations of many complex real-world systems. Analyzing various structural properties of and dynamics on such networks reveal useful insights about the real-world systems (Newman 2002). Assortative edge switch is an important problem in the analysis of such networks and has many real-world applications. An edge switch (or edge swap, edge flip, edge shuffle, edge rewiring) is an operation on a network where two edges are selected randomly and the end vertices are swapped with each other. More formally, it selects two edges (a,b) and (c,d) uniformly at random and replaces them with edges (a,d) and (c,b), respectively. We refer to this operation as the *regular edge switch*. It is easy to see that this operation preserves the degree of each vertex. It is repeated either as many times as required or a specific criterion is satisfied. Edge switch can be used in the generation of random networks with a given degree sequence (Cooper et al. 2007), independent realizations of graphs with a prescribed joint degree distribution using a Markov chain Monte Carlo approach (Ray et al. 2012), modeling and studying various dynamic networks (Feder et al. 2006) and in many other network analytic problems.

Many variations of the edge switch problem (Cooper et al. 2007, Feder et al. 2006, Ray et al. 2012) have been studied. In this paper, we present efficient sequential and parallel algorithms for such a variant, referred to as the *assortative edge switch*, which is an operation on a labeled network, where each vertex u has an associated label $L(u)$. Such labels can be discrete characteristics (e.g., language, race, and gender in social networks) or scalar properties (e.g., age and degree) of the vertices. An assortative edge switch operation selects two edges (a,b) and (c,d) randomly, and replaces them with edges (a,d) and (c,b), respectively, if $L(a) = L(c)$ and $L(b) = L(d)$. For each vertex u, the degree, as well as the distribution of the labels of the adjacent vertices of u, remains invariant under an assortative edge switch process.

An assortative edge switch operation preserves the *assortative mixing* of a given network, which is a fundamental network feature measuring the tendency of the vertices to associate with similar or dissimilar vertices and is quantified by a metric named the *assortative coefficient* (Newman 2003). In other words, assortativity measures the correlation of vertices based on the vertex labels. Newman (2002) showed that assortative mixing is a pervasive phenomenon found in many real-world networks and has a profound impact on the structural properties and functionalities of the networks. For instance, social (e.g., co-authorship) networks exhibit positive assortativity—more association among the similar types of vertices—whereas technological (e.g., Internet and WWW) and biological (e.g., food-web) networks show negative assortativity—more association among the dissimilar types of vertices (Newman 2002). Assortative edge switch can be used to assess and analyze the sensitivity of mixing patterns and network structural properties on dynamics over a network, such as disease dynamics over a social contact network (Eubank et al. 2010). Random network models often do not capture many structural properties (e.g., assortative mixing) of real-world networks; as a result, to be more realistic, modeling random networks with a prescribed assortative coefficient has gained popularity in the research community (Milo et al. 2003). Assortative edge switch can be combined with regular edge switch process to generate random networks with a prescribed assortative coefficient from a given network (Xulvi-Brunet and Sokolov 2004).

The recent growth of real-world data due to the rapid progression of science and technology poses significant challenges towards efficient processing of massive networks. Dealing with these huge amounts of data efficiently and effectively motivates the need for parallel computing. NetworkX (Hagberg et al. 2008) has a sequential implementation of regular edge switch; however, it does not have an implementation of assortative edge switch. A parallel algorithm for regular edge switch has been presented in (Bhuiyan et al. 2014). However, no effort was given to design a parallel algorithm for assortative edge switch in a network. Although the algorithm by Bhuiyan et al. (2014) can be applied to perform assortative edge switch operations in a network, it can lead to a slow and inefficient algorithm. To perform an edge switch operation in (Bhuiyan et al. 2014), the edges are selected randomly from the entire network, irrespective of the vertex labels, which can result in many failed attempts for an assortative edge switch operation due to dissatisfying the constraint on the vertex labels. As a result, we need a completely new and efficient algorithmic approach.

Our Contributions. In this paper, we first present a sequential algorithm for assortative edge switch; then we present a parallel algorithm based on our sequential algorithm. The dependencies among successive assortative edge switch operations and the requirement of keeping the network simple as well as maintaining the same assortativity during the assortative edge switch process pose significant challenges in designing a parallel algorithm. Moreover, achieving a good speedup through a well-balanced load distribution among the processors seems to be a non-trivial challenge for this problem. For various network size and diverse distribution of the labels, the parallel algorithm provides speedup ranging between 68 and 772 with 1024 processors. To our knowledge, they are the first efficient sequential and parallel algorithms for the problem.

Organization. The remainder of the paper is organized as follows. The preliminaries and datasets used in the paper are briefly described in Section 2. The problem of assortative edge switch and the sequential algorithm are discussed in Section 3. We present the parallel algorithm along with the performance analysis in Section 4. Finally, we conclude in Section 5.

2 PRELIMINARIES AND DATASETS

Below are the notations, definitions, datasets, and computation model used in this paper.

2.1 Notations and Definitions

We are given a simple, labeled network $G = (V, E, L)$, where V is the set of vertices, E is the set of edges, and $L : V \to \mathbb{N}_0$ is the label function. A *simple network* is an undirected network having no self-loops or parallel edges. A *self-loop* is an edge from a vertex to itself. *Parallel edges* are two or more edges connecting the same pair of vertices. There are a total $n = |V|$ vertices with vertex ids $0, 1, 2, \ldots, n-1$ and $m = |E|$ edges in G. Each vertex $u \in V$ has an associated label $L(u)$. There are a total of ℓ distinct vertex labels with label ids $0, 1, 2, \ldots, \ell - 1$, i.e., for each $u \in V$, $L(u) \in [0, \ell - 1]$. For an edge $(u, v) \in E$, we say u and v are *neighbors* of each other. The *adjacency list* of a vertex u contains all the neighbors of u and is denoted by $N(u)$, i.e., $N(u) = \{v \in V | (u, v) \in E\}$. The *degree* of u is $d_u = |N(u)|$. We use K, M, and B to denote thousands, millions, and billions, respectively; e.g., $1B$ stands for one billion. For the parallel algorithm, let there be P processors with ranks $0, 1, 2, \ldots, P-1$, where P_i denotes the processor with rank i.

Definition 1. *An* **edge switch** *operation selects two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ randomly from E and replaces them with new edges $e_3 = (u_1, v_2)$ and $e_4 = (u_2, v_1)$, if the resultant network remains simple.*

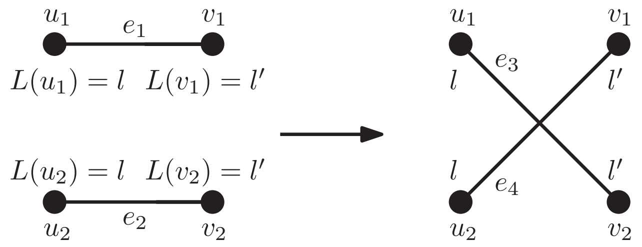


Figure 1: An assortative edge switch operation replaces two randomly selected edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ by new edges $e_3 = (u_1, v_2)$ and $e_4 = (u_2, v_1)$, if $L(u_1) = L(u_2)$ and $L(v_1) = L(v_2)$.

Definition 2. *An* **assortative edge switch** *operation imposes an extra constraint on the labels of the end vertices of the two selected edges in addition to the regular edge switch constraints. That is, it randomly selects two edges $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ from E, and replaces them by new edges $e_3 = (u_1, v_2)$ and $e_4 = (u_2, v_1)$ (see Figure 1), if the following constraints are satisfied.*

- **Simple network:** *It does not create self-loops or parallel edges, i.e., $u_1 \neq v_2$, $u_2 \neq v_1$, $u_1 \notin N(v_2)$, $v_2 \notin N(u_1)$, $u_2 \notin N(v_1)$, and $v_1 \notin N(u_2)$.*
- **Vertex labels:** *The two edges have the same end vertex labels, i.e., $L(u_1) = L(u_2)$, $L(v_1) = L(v_2)$.*

Definition 3. *Some edges of the given network G are changed (or visited) due to assortative edge switch operations, and some edges do not participate in any such operations and remain unchanged (or unvisited). We define the* **visit rate** *(x) as the fraction of edges of G that have been changed by a sequence of assortative edge switch operations, i.e., $x = \frac{m'}{m}$, where m' is the number of edges changed due to assortative edge switches. To achieve a given visit rate x, the expected number of assortative edge switch operations τ to be performed in G is $\tau = \frac{1}{2} m \ln m$ for $x = 1$, and $\tau = -\frac{1}{2} m \ln(1-x)$ for $0 < x < 1$ (Bhuiyan et al. 2014).*

Table 1: Datasets used in the experiments. K, M, and B denote thousands, millions, and billions, respectively.

Network	**Type of network**	**Vertices**	**Edges**	**Bins**		**Assort. edge switch**
				Deg. assort.	**Age assort.**	**(Visit rate = 1.0)**
New York	Social Contact	17.88**M**	480.1**M**	85.7**K**	4186	4.80**B**
Los Angeles	Social Contact	16.23**M**	459.3**M**	83.1**K**	4186	4.58**B**
Miami	Social Contact	2.09**M**	51.50**M**	66.1**K**	4186	457.2**M**
Sweden	Social Contact	9.46**M**	406.2**M**	83.2**K**	-	4.03**B**
Orkut	Social	3.07**M**	117.2**M**	2.46**M**	-	1.09**B**
Facebook	Social	3.10**M**	23.67**M**	378**K**	-	200.9**M**
LiveJournal	Social	4.80**M**	42.85**M**	802**K**	-	376.5**M**
ErdosRenyi	Erdős-Rényi	1.00**M**	500.00**M**	36.62**K**	-	5.0**B**
SmallWorld	Small World	4.80**M**	48.00**M**	161	-	424.5**M**
PA	Pref. Attach.	100.0**M**	1.00**B**	1.58**M**	-	10.36**B**

2.2 Datasets and Computation Model

We use both artificial and real-world networks for the experiments. A summary of the networks is given in Table 1. New York (NY), Los Angeles (LA), Miami, and Sweden are synthetic, yet realistic social contact networks (Barrett et al. 2009). Each vertex represents a person in the corresponding city or country and has an associated label denoting the age of the individual, and each edge represents any 'physical' contact between two persons within a 24 hour time period. Orkut is an online social network, Facebook is an anonymized Facebook friendship network, and LiveJournal is a social network blogging site (Abdelhamid et al. 2012). The SmallWorld, ErdosRenyi, and PA networks are random networks generated using the Watts-Strogatz small world network (Watts and Strogatz 1998), Erdős-Rényi network (Bollobás 1998), and Preferential Attachment network (Barabási and Albert 1999) models, respectively. For all the networks, we perform *degree-assortative* edge switch operations, where the degree of each vertex is considered as its label. In addition, we perform *age-assortative* edge switch operations on the social contact networks of Miami, NY, and LA, where each person's age is considered as its label. The age information for the other networks is either inappropriate or unavailable.

We develop algorithms for distributed memory parallel systems, where each processor has its own local memory. The processors do not have any shared memory and can communicate with each other by exchanging messages.

3 A SEQUENTIAL ALGORITHM

We are given a network $G = (V, E, L)$ and the number of assortative edge switch operations τ to be performed. A naïve approach to perform an assortative edge switch operation is selecting two edges (u_1, v_1) and (u_2, v_2) uniformly at random from E and swapping the end vertices of the edges if the constraints are satisfied. If any of the constraints is not satisfied, the selected pair of edges is discarded and a new pair is selected. For a large and relatively sparse network, the number of such discarded attempts (or pairs of edges) due to dissatisfying the constraint of keeping the network simple is almost negligible; however, for the constraint on the vertex labels, the number of discarded attempts can be very large, as shown in Table 2. For example, to perform 10K degree-assortative edge switch operations on the LA network, the numbers of discarded attempts due to the constraints on the simple network and vertex labels are 1 and 179.4M, respectively. Assume that there are $\ell = 100$ different labels uniformly distributed among the vertices of a given network, i.e., for any label i, the number of vertices with label i is $\frac{n}{100}$. Then the probability of randomly selecting two edges (u_1, v_1) and (u_2, v_2) satisfying the constraint on the vertex labels is $(\frac{1}{100})^2$, which is very

Table 2: Number of discarded attempts (due to dissatisfying the two constraints) to perform 10K age- and degree-assortative edge switch operations on different networks.

Constraints	**Miami-Age**	**LA-Deg**	**Facebook**	**LiveJournal**	**ErdosRenyi**
Simple network	2	1	148	34	23
Vertex labels	30.3**M**	179.4**M**	617.2**M**	165.2**M**	125.4**M**

small. As a result, the number of discarded attempts can be very large, and it can make the algorithm slow. To deal with this difficulty, we present an efficient sequential algorithm using a new and efficient algorithmic approach.

3.1 An Efficient Sequential Algorithm

Let us denote a *bin* Z_{ij} to be the set of all edges having the same end vertex labels (i,j), where $0 \le i,j \le \ell - 1$, i.e., for an edge $(u,v) \in E$, if $L(u) = i$ and $L(v) = j$, then $(u,v) \in Z_{ij}$. For an undirected network, $Z_{ij} = Z_{ji}$, and we use Z_{ij} such that $j \le i$. Note that the bins are disjoint and $\bigcup_{j \le i} Z_{ij} = E$. For convenience, we relabel the bins from two indices (i,j) to a single bin number k. Let b_k be the new label of the bin Z_{ij}, where $k = \frac{i(i+1)}{2} + j$. Assume that there are Y such bins and let us denote them as $b_0, b_1, \ldots, b_{Y-1}$; there can be at most $Y = \binom{\ell}{2} + \ell = O(\ell^2)$ such bins. The *size* of a bin b_i is the number of edges in b_i and is denoted by m_i, i.e., $m_i = |b_i|$. Then the number of possible pairs of edges in b_i is $a_i = {m_i}^2$. Note that the set of edges in a bin b_i changes dynamically during the course of an assortative edge switch process, although m_i remains invariant throughout the process.

First, the algorithm constructs the bins b_i from G. Then an assortative edge switch operation is performed as follows: (*i*) a bin b_i is chosen randomly, where the probability of selecting b_i is $q_i = \frac{a_i}{\sum_{j=0}^{Y-1} a_j}$, (*ii*) a pair of edges is selected uniformly at random from b_i, and (*iii*) the end vertices of the edges are swapped with each other. This operation is repeated until τ pairs of edges are switched. Note that this algorithm guarantees that both of the edges for an assortative edge switch operation are always selected from the same bin irrespective of how many bins there are, thus overcoming the drawback of the naïve approach. A pseudocode of the algorithm is given in Figure 2.

```
1: Partition $E$ into minimum number of disjoint bins $b_0, b_1, \ldots, b_{Y-1}$, where for any $i$ and for any
   pair of edges $(u_1,v_1),(u_2,v_2) \in b_i$, $L(u_1) = L(u_2)$ and $L(v_1) = L(v_2)$.
2: for $k = 1$ to $\tau$ do
3:     $b_i \leftarrow$ a random bin in $[b_0, b_{Y-1}]$ with a probability of $q_i = \frac{a_i}{\sum_{j=0}^{Y-1} a_j}$
4:     $(u_1,v_1),(u_2,v_2) \leftarrow$ two uniform random edges in $b_i$
5:     if $u_1 = v_2$, $u_2 = v_1$, $u_1 \in N(v_2)$, or $u_2 \in N(v_1)$ then
6:         continue
7:     Replace $(u_1,v_1)$ and $(u_2,v_2)$ by $(u_1,v_2)$ and $(u_2,v_1)$
```

Figure 2: A sequential algorithm for assortative edge switch.

Theorem 1. *The time complexity of the sequential algorithm is* $O(m + Y + \tau \log d_{max})$.

Proof. Partitioning the set of edges E into the Y bins (line 1 in Figure 2) takes $O(m+Y)$ time as initializing the bins takes $O(Y)$ time and the bins are constructed from E in $O(m)$ time. The adjacency list of a vertex u can be maintained using a balanced binary tree, and searching (for parallel edges) and updating such a tree takes $O(\log d_u)$ time. Hence, an assortative edge switch operation (lines $3-7$) can be performed

Table 3: Runtime comparison of our sequential algorithm and the naïve approach on various networks. Experiments performed with a visit rate of 0.01.

Network	**Runtime (min.)**		**Faster by a**
	Naïve approach	**Our algo.**	**factor of**
Miami-Age	8.8	0.14	62.86
LA-Deg	929	1.88	494.1
Facebook	69.1	0.08	863.75
LiveJournal	48.87	0.15	325.8
ErdosRenyi	342.5	2.22	154.3

in $O(\log d_{max})$ time, where $d_{max} = \max_u d_u$ and τ such operations (lines $2-7$) take $O(\tau \log d_{max})$ time. Therefore, the time complexity of the algorithm is $O(m+Y+\tau \log d_{max})$. ■

Theorem 2. *The space complexity of the sequential algorithm is $O(m+Y)$.*

Proof. Storing all the m edges into the Y bins takes $O(m+Y)$ space. ■

3.2 Performance Evaluation of the Sequential Algorithm

Table 3 demonstrates the runtime comparison of our algorithm with the naïve approach. We use current calendar time as a random seed and a visit rate of 0.01 for the experiments since the naïve approach takes a large amount of time with a visit rate of 1.0. Our algorithm shows very good overall performance, e.g., for degree-assortative edge switch on the LA network, our algorithm is 494 times faster than the naïve approach.

4 THE PARALLEL ALGORITHM

A parallel algorithm for regular edge switch is presented in (Bhuiyan et al. 2014); however, this algorithm can be very slow for assortative edge switch for the same reasons as explained in Section 3. In this section, we present a novel parallel algorithm for assortative edge switch, which is based on our sequential algorithm, as shown in Figure 2. Recall that the sequential algorithm selects both of the edges for an assortative edge switch operation from the same bin. Hence, assortative edge switch operations in a bin are independent of the other bins. The parallel algorithm exploits this property to perform simultaneous assortative edge switch operations in parallel in different bins. The bins are distributed among the processors such that the computation load distribution is well-balanced. If a bin is very large compared to the other bins, the algorithm might need to partition and distribute the bin among multiple processors, which is discussed later in Section 4.3. For now, assume that each bin is entirely assigned to a single processor.

4.1 The Parallel Algorithm with Each Bin Assigned to a Single Processor

The parallel algorithm should distribute the bins to processors such that the computation cost is equally distributed among the processors. Therefore, we need to estimate the computation cost associated with each bin, which is the number of assortative edge switch operations performed in a bin b_i. It raises the question of how many assortative edge switch operations among the total τ operations are performed in b_i? Let us denote X_i be the number of assortative edge switch operations performed in b_i by the sequential algorithm. Recall that to perform an assortative edge switch operation, the sequential algorithm randomly selects a bin b_i, and then chooses two edges randomly from b_i. Hence, a sequential algorithm does not need to know X_i in advance. However, in the parallel algorithm, all of the processors perform simultaneous assortative edge switch operations in parallel and different processors may need to work on different bins at the same time. As a result, for each i, X_i needs to be determined in advance for the parallel algorithm. It is easy to see that

```
1: for $i = 0$ to $P-1$ do
2:     $W_i \leftarrow 0$
3:     $B_i \leftarrow \emptyset$
4: Sort the bins in non-increasing order of $X_j$
5: Assume that $X_0 \geq X_1 \geq \ldots \geq X_{Y-1}$
6: for $j = 0$ to $Y-1$ do
7:     Let $P_i$ be the processor with rank $i = \arg\min_k W_k$
8:     $B_i \leftarrow B_i \bigcup \{b_j\}$
9:     $W_i \leftarrow W_i + X_j$
```

Figure 3: A load balancing algorithm assigning the Y bins to the P processors.

```
1: for each bin $b_j \in B_i$ do
2:     for $k = 1$ to $X_j$ do
3:         $(u_1,v_1),(u_2,v_2) \leftarrow$ two uniform random edges in $b_j$
4:         if $u_1 = v_2$, $u_2 = v_1$, $u_1 \in N(v_2)$, or $u_2 \in N(v_1)$ then
5:             continue
6:         Replace $(u_1,v_1)$ and $(u_2,v_2)$ by $(u_1,v_2)$ and $(u_2,v_1)$
```

Figure 4: A processor P_i performing assortative edge switch operations in the parallel algorithm.

the random variables X_i are the multinomial random variables generated by a multinomial distribution with parameters $(\tau, q_0, q_1, \ldots, q_{Y-1})$, that is,

$$\langle X_0, X_1, \ldots, X_{Y-1} \rangle \sim M(\tau, q_0, q_1, \ldots, q_{Y-1}) \tag{1}$$

where τ is the number of assortative edge switch operations and $q_i = \frac{a_i}{\sum_{j=0}^{Y-1} a_j}$ is the probability of selecting a bin b_i. The time complexity of the best known sequential algorithm, known as the *conditional distributed method* (Davis 1993), for generating multinomial random variables is $\Theta(\tau)$. To have an efficient parallel algorithm for assortative edge switch, we need a parallel algorithm for generating multinomial random variables. We use the algorithm presented in (Bhuiyan et al. 2014), which has a runtime of $O\left(\frac{\tau}{P} + Y \log P\right)$. Each processor independently computes the multinomial distribution of $\frac{\tau}{P}$ and then the results are aggregated by exploiting a property of the multinomial distribution. An overview of the parallel algorithm is as follows:

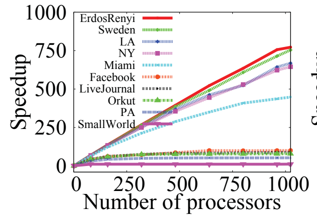
- ***Step 1:*** Generate multinomial random variables X_i in parallel with parameter $(\tau, q_0, q_1, \ldots, q_{Y-1})$ to estimate the computation cost associated with each bin.
- ***Step 2:*** Using the estimated costs X_i, partition the bins $b_0, b_1, \ldots, b_{Y-1}$ among the P processors such that the computation load distribution is well-balanced.
- ***Step 3:*** Each processor P_i simultaneously performs assortative edge switch operations in parallel in the bins assigned to it.

Now we describe the last two steps of the algorithm.

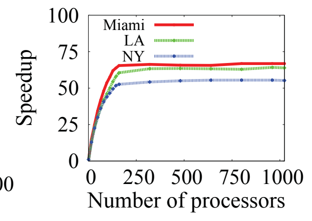
Partitioning (Step 2). Let B_i be the set of bins, $Y_i = |B_i|$ be the number of bins and $M_i = \sum_{b_j \in B_i} m_j$ be the number of edges assigned to a processor P_i. Then the *workload* (or *load*) W_i in P_i is the summation of the computation costs associated with the bins in B_i, i.e., $W_i = \sum_{b_j \in B_i} X_j$. Let W be the *maximum load* among all of the processors, i.e., $W = \max_i W_i$. Now the goal is to distribute the Y bins among the P processors such that the maximum load W is minimized. Finding an assignment of the bins for the optimum solution is an NP-hard problem (Kleinberg and Tardos 2006). The best known approximation algorithm for this problem is presented in (Kleinberg and Tardos 2006), which has an approximation ratio of 1.5. This greedy algorithm sorts the bins in non-increasing order of the loads. To do so, we use a parallel version of quick sort (Grama 2003). Then the algorithm makes one pass over the sorted bins to assign each bin to a processor having the minimum load at the time of the assignment, as shown in Figure 3.

Switching Edges (Step 3). Each processor P_i constructs the bins in B_i and then simultaneously performs W_i assortative edge switch operations in parallel, as shown in Figure 4. The program terminates when all of the processors complete their assortative edge switch operations.

Theorem 3. *The time complexity in each processor P_i is $O(\frac{\tau}{P} + Y \log P + \frac{Y}{P} \log Y + \log^2 P + P + Y_i + M_i + W_i \log d_{max})$.*

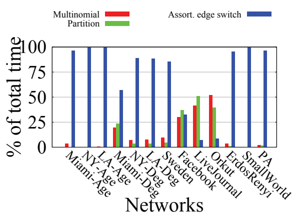


(a) Degree-assortative

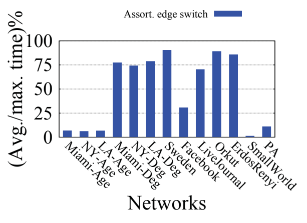


(b) Age-assortative

Figure 5: Strong scaling of the parallel algorithm.

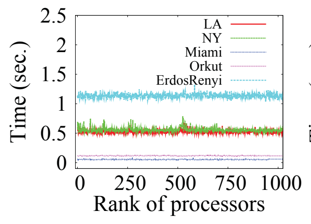


(a) Time taken by individual steps of the algorithm

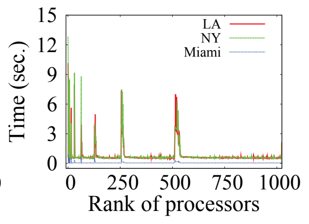


(b) Ratio of the avg. and max. time in the third step

Figure 6: Time at different steps of the algorithm.

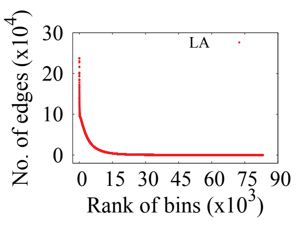


(a) Degree-assortative

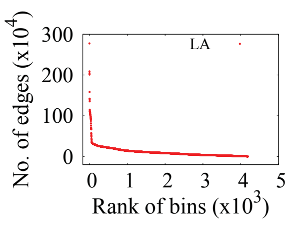


(b) Age-assortative

Figure 7: Execution time of the individual processors in the third step of the algorithm.



(a) Degree-assortative



(b) Age-assortative

Figure 8: Distribution of edges among the bins for the LA network.

Proof. In step 1, multinomial random variables are generated in parallel in $O\left(\frac{\tau}{P} + Y\log P\right)$ time. In step 2, the parallel version of quick sort and the assignment of the bins take $O\left(\frac{Y}{P}\log Y + \log^2 P\right)$ and $O\left(P + Y\log P\right)$ time, respectively. In step 3, each processor P_i constructs the Y_i bins in $O\left(Y_i + M_i\right)$ time and then performs the assortative edge switch operations in $O\left(W_i \log d_{max}\right)$ time. ■

Theorem 4. *The space complexity in each processor P_i is $O\left(Y_i + M_i\right)$.*

Proof. Each processor P_i stores all the M_i edges in Y_i bins using $O\left(Y_i + M_i\right)$ space. ■

4.2 Performance Analysis of the Parallel Algorithm with Each Bin Assigned to a Single Processor

In this section, we analyze the performance of the parallel algorithm. Table 1 provides a summary of the number of bins and assortative edge switch operations performed on the ten networks. We use a HPC cluster consisting of 64 compute nodes. Each node has a dual-socket Intel Sandy Bridge E5-2670 2.60GHz 8-core processor (16 cores per node) with 64GB memory. The algorithms are developed with MPICH2 (v1.9), optimized for Qlogic QDR Infiniband cards.

4.2.1 Strong Scaling

Figure 5a and 5b illustrate the strong scaling performance for degree- and age-assortative edge switch, respectively, on various networks. The algorithm achieves a speedup between 12 and 772 with 1024 processors. For some networks, the speedup is poor compared to the other networks. Next, we investigate the load distribution among the processors to understand the reason for this poor performance.

4.2.2 Load Distribution

First, we measure the time taken by each of the three steps (step 1: multinomial, step 2: partitioning, step 3: assortative edge switch) of the algorithm with 1024 processors, as shown in Figure 6a. The assortative edge switch step takes the largest amount of time among the three steps for all the networks, except Facebook, LiveJournal, and Orkut, in which, the number of bins Y is significantly higher than that of the other networks (see Table 1). Therefore, generating the multinomial random variables and partitioning the bins take more time for these three networks. Unlike the first two steps, where every processor takes almost an equal amount of time, the execution time of the individual processors in the third step can vary significantly because of a poor load distribution. Figure 6b demonstrates the ratio of T_{avg} and T_{max}, where T_{avg} is the average execution time of the processors in the assortative edge switch step and T_{max} is the maximum time among them. The maximum value of the ratio is 100% for a perfectly-balanced load distribution in the ideal case. A higher value implies a well-balanced load distribution, whereas a lower value indicates a poor load distribution among the processors. We observe a well-balanced load distribution for degree-assortative edge switch on the Miami, NY, LA, Sweden, LiveJournal, Orkut, and ErdosRenyi networks. In contrast, we observe a poor load distribution for age-assortative edge switch on the Miami, NY and LA networks and degree-assortative edge switch on the PA and SmallWorld networks. For the PA network, the poor load distribution in the third step causes the step to take the largest amount of time despite having a large number of bins, which is in contrast to the scenario for the Facebook, LiveJournal, and Orkut networks. For the Facebook network, a few processors contain many small-size bins (with a few edges) and the number of discarded attempts in these bins are very high, yielding a larger execution time in these processors (hence, T_{max} is large). As a result, we observe a low ratio despite each processor performing roughly an equal number of assortative edge switch operations and the low ratio is not a consequence of load balancing. The observations are further supported by Figures 7a and 7b, which show the individual execution time of the processors in the third step of the algorithm. For a better understanding, we analyze the load distribution in detail for the LA network.

Figures 8a and 8b illustrate the distribution of the sizes of the bins for degree- and age-assortative edge switch, respectively, on the LA network. The distribution for age-assortative edge switch is highly skewed, having a few very large bins and many small bins, which in turn makes a poor load distribution among the processors, as shown in Figure 7b. A few processors containing the larger bins are taking significantly more time than the processors containing the smaller bins, which results in a low speedup. On the other hand, for degree-assortative edge switch, there is a good number of both larger and smaller bins among the total 83.1K bins and the differences between the larger and smaller bins are substantially smaller than that of the age-assortative counterpart. The algorithm assigns a few larger bins along with many smaller bins to each processor and consequently exhibits a well-balanced load distribution, as shown in Figure 7a, thus resulting in a good speedup. Note that if the number of bins is less than the number of processors, i.e., $Y < P$, then some processors remain idle in the third step of the computation; and we observe this phenomenon for the SmallWorld network. To deal with the poor load distribution, we present a parallel algorithm with an improved load balancing scheme in the next section.

4.3 The Parallel Algorithm with Improved Load Balancing

As we discussed earlier, some bins can have higher computation costs, i.e., the number of assortative edge switch operations X_i performed in a bin b_i can be significantly larger than $\frac{\tau}{P}$, which can cause a poor load distribution. For a better load balancing, such a bin may need to be partitioned and distributed among multiple processors. Let Δ be some *threshold* such that Δ is larger than $\frac{\tau}{P}$. We call a bin *large* if $X_i \geq \Delta$, and *small*, otherwise. Let Q be the total number of processors assigned for the large bins. First, we explain how to perform assortative edge switch operations in a single large bin with multiple processors.

Assume that a large bin b_i is partitioned among the processors $P_x, P_{x+1}, \ldots, P_y$, where $x \leq j,k,l \leq y$. The bin b_i is partitioned such that a subset of vertices, having consecutive vertex ids, along with their incident edges in b_i are assigned to a partition and each such partition contains almost an equal number of edges. A vertex u's *partial* adjacency list in b_i, i.e., $N_i(u) = \{v \in V | (u,v) \in b_i\}$, entirely belongs to a unique partition. Then each processor P_j performs simultaneous assortative edge switch operations in parallel. Assortative edge switch in a bin is similar to the regular edge switch because the end vertices of the edges in a bin have the same labels. A parallel algorithm for regular edge switch has been presented in (Bhuiyan et al. 2014). We use this algorithm to switch the edges in a large bin. A summary of an assortative edge switch operation performed by a processor P_j is as follows. P_j selects an edge (u_1, v_1) randomly from its own partition. The other edge (u_2, v_2) is chosen in two steps: (*i*) P_j selects a processor P_k with a probability proportional to the number of edges belonging to P_k, and (*ii*) P_j requests P_k to select (u_2, v_2) randomly from its partition. Then P_j and P_k work together to check whether switching the edges (u_1, v_1) and (u_2, v_2) creates any loop or parallel edge. If it creates any loop or parallel edge, the selected pair of edges is discarded, and a new pair is chosen by P_j. Otherwise, P_j and P_k work together to update (u_1, v_1) and (u_2, v_2) by (u_1, v_2) and (u_2, v_1), respectively. In fact, P_j and P_k may require updating an edge in another processor P_l ($P_j \neq P_l \neq P_k$). The details can be found in (Bhuiyan et al. 2014).

Now, we discuss how to determine Δ and Q. Apparently, it seems that any bin b_i with $X_i > \frac{\tau}{P}$ needs to be partitioned and distributed among multiple processors. However, partitioning a bin incurs communication and synchronization overhead due to the need for exchanging messages among multiple processors even for a single edge switch operation. Thus, we partition a bin among multiple processors only when X_i is significantly larger than $\frac{\tau}{P}$, i.e., when the gain achieved by partitioning a bin is larger than the communication and synchronization cost incurred by partitioning the bin. We assume $\Delta = \alpha \times \frac{\tau}{P}$ for some constant $\alpha > 1$. Similarly, to perform X_i assortative edge switch operations in a large bin b_i, we should ideally assign $\lceil \frac{X_i}{\tau/P} \rceil$ processors for b_i. However, due to communication overhead, we need to assign a larger number of processors. Hence, we assign $Q_i = \lceil \beta \times X_i \times \frac{P}{\tau} \rceil$ processors for a large bin b_i for some constant $\beta > 1$. Then $Q = \sum_i Q_i$ and the number of processors assigned for the small bins is $S = P - Q$. The performance of the algorithm greatly depends on Δ and Q, thusly on α and β. We experimented with many different types of networks and find that for $\alpha \in [2.4, 2.75]$ and $\beta \in [12, 15]$, the algorithm exhibits good performance, which is very close to the optimal performance, as shown in the next section.

4.4 Performance Analysis of the Parallel Algorithm with Improved Load Balancing

Figure 9 shows the strong scaling performance of the parallel algorithm and Figure 10 demonstrates a comparison of the speedup improvement. The algorithm achieves a speedup of 277 for age-assortative edge switch on the Miami network with 1024 processors, which is a four-fold improvement achieved by a well-balanced load distribution among the processors, as shown in Figure 11. Next, we analyze the effect of α and β (thus Δ and Q) on the speedup.

Figure 12 shows how the speedup varies with different values of α and β for age-assortative edge switch on the Miami network with 1024 processors. For a fixed value of β (say $\beta = 14$) and with the increase of α, more large-size bins are getting partitioned among the same number of processors, yielding a speedup increase up to some value of α, referred to as *optimal* α (optimal $\alpha = 2.5$ for $\beta = 14$), beyond which the speedup starts decreasing because of the increase of communication and synchronization overhead incurred by the increasing number of bins partitioned. We observe a similar pattern for a fixed value of α and with the increase of β as well. For lower values of β, less number of processors are working on the large bins. As a result, the execution times of the Q processors are higher than that of the S processors working on the small bins. This is further illustrated in Figure 13, which shows the execution time of the individual processors with varying β and a fixed $\alpha = 2.5$. We observe a pattern of many horizontal flat segments, where each horizontal segment is the time taken by the processors working on the same large bin. With the increase of β, thusly Q, the amount of work for each of the Q processors decreases, whereas the times taken by the S processors increase as fewer processors are dealing with the small bins. The highest speedup is achieved when the maximum execution time among the Q processors is somewhat balanced with that of the S processors. We observe similar phenomena for the other networks as well and recommend using a $\alpha \in [2.4, 2.75]$ and $\beta \in [12, 15]$, for which the algorithm achieves a good speedup.

In principle, the parallel algorithm is designed such that it stochastically produces the same effect (by using the multinomial distribution) on a network as the sequential algorithm would do. We also experimentally verify this by showing that the average clustering coefficient and the average shortest path distance of a network change similarly with assortative edge switches by the sequential and parallel algorithms (see Fig-

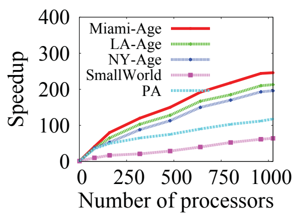


Figure 9: Strong scaling of the parallel algorithm with improved load balancing.

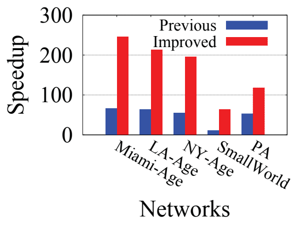


Figure 10: A comparison of speedup improvement with 1024 processors.

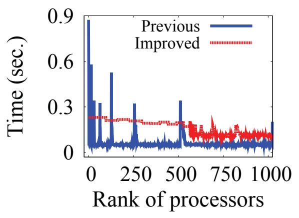


Figure 11: A comparison of runtime of the individual processors for Miami-Age.

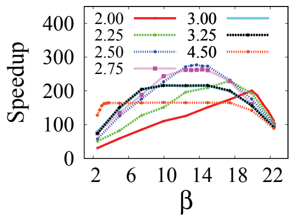


Figure 12: Speedup with increasing β for Miami-Age using different values of α.

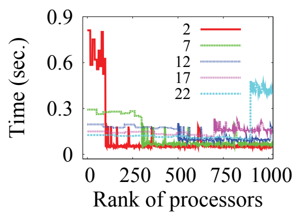


Figure 13: Runtime of the individual processors with different values of β for Miami-Age using a fixed $\alpha = 2.5$.

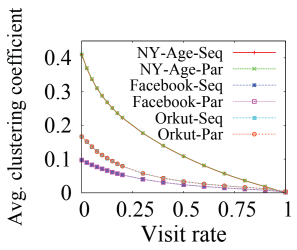


Figure 14: Average clustering coefficient changes similarly with assort. edge switches by the sequential and parallel algorithms.

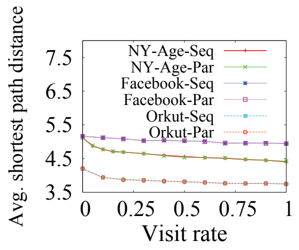


Figure 15: Average shortest path distance changes similarly with assort. edge switches by the sequential and parallel algorithms.

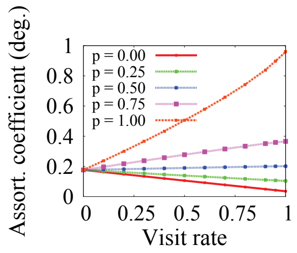


Figure 16: Varying degree-assort. coefficient by age-assort. edge switch on the Miami network through a parameter $p \in [0, 1]$.

ure 14 and 15). We use the NY, Orkut, and Facebook networks and vary the visit rate from 0.025 to 1. For both properties, the changes by the sequential and parallel algorithms are very similar; in fact, they overlap with each other, and it is difficult to distinguish them in the figures. We also demonstrate how assortative edge switch can be used to generate random networks by keeping one assortative coefficient A_1 invariant and varying another assortative coefficient A_2 to a desired level through a parameter p $(0 \leq p \leq 1)$. Xulvi-Brunet and Sokolov (2004) proposed one such algorithm, where with probability p, an edge switch operation connects the two higher degree vertices with an edge and the two lower degree vertices with another edge. With probability $(1 - p)$, the edges are switched randomly. Figure 16 shows how the degree assortative coefficient changes for different value of p with the age-assortative edge switch process on the Miami network.

5 CONCLUSION

We have developed efficient sequential and parallel algorithms for assortative edge switch in massive networks. They can be used to study the sensitivity of network topology on the dynamics over a network as well as to generate network perturbations of a given network by maintaining the same degree sequence and assortative coefficient.

ACKNOWLEDGMENTS

This work has been partially supported by DTRA CNIMS Contract HDTRA1-11-D-0016-0001, DTRA Grant HDTRA1-11-1-0016, NSF NetSE Grant CNS-1011769, NSF SDCI Grant OCI-1032677, NIH MIDAS Grant 5U01GM070694-11, NSF DIBBs Grant ACI-1443054, and NSF Big Data Grant IIS- 1633028.

REFERENCES

Abdelhamid, S. E., R. Alo, S. Arifuzzaman, P. Beckman, M. H. Bhuiyan et al. 2012. “CINET: A cyberinfrastructure for network science”. In *Proceedings of the 8th International Conference on E-Science (e-Science)*, pp. 1–8.

Barabási, A.-L., and R. Albert. 1999. "Emergence of scaling in random networks". *Science* vol. 286 (5439), pp. 509–512.

Barrett, C. L., R. J. Beckman, M. Khan et al. 2009. "Generation and analysis of large synthetic social contact networks". In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pp. 1003–1014.

Bhuiyan, H., J. Chen, M. Khan, and M. V. Marathe. 2014. "Fast parallel algorithms for edge-switching to achieve a target visit rate in heterogeneous graphs". In *Proceedings of the 43rd International Conference on Parallel Processing (ICPP)*, pp. 60–69.

Bollobás, B. 1998. "Random graphs". In *Modern Graph Theory*, pp. 215–252. Springer.

Cooper, C., M. Dyer, and C. Greenhill. 2007. "Sampling regular graphs and a peer-to-peer network". *Combinatorics, Probability and Computing* vol. 16 (4), pp. 557–593.

Davis, C. S. 1993. "The computer generation of multinomial random variates". *Computational Statistics & Data Analytics* vol. 16 (2), pp. 205–217.

Eubank, S., A. Vullikanti, M. Khan et al. 2010. "Beyond degree distributions: Local to global structure of social contact graphs". In *Proceedings of the Third International Conference on Social Computing, Behavioral Modeling, and Prediction*, pp. 1–1.

Feder, T., A. Guetz, M. Mihail, and A. Saberi. 2006. "A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks". In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 69–76.

Grama, A. 2003. *Introduction to parallel computing*. Pearson Education.

Hagberg, A., P. Swart, and D. Schult. 2008. "Exploring network structure, dynamics, and function using NetworkX". In *Proceedings of the 7th Python in Science Conference (SciPy)*, pp. 11–15.

Kleinberg, J., and É. Tardos. 2006. *Algorithm design*. Pearson Education.

Milo, R., N. Kashtan, S. Itzkovitz, M. E. Newman, and U. Alon. 2003. "On the uniform generation of random graphs with prescribed degree sequences". *arXiv preprint cond-mat/0312028*.

Newman, M. E. 2002. "Assortative mixing in networks". *Physical Review Letters* vol. 89 (20), pp. 208701.

Newman, M. E. 2003. "Mixing patterns in networks". *Physical Review E* vol. 67 (2), pp. 026126.

Ray, J., A. Pinar, and C. Seshadhri. 2012. "Are we there yet? When to stop a Markov chain while generating random graphs". In *Proceedings of the 9th Workshop on Algorithms and Models for the Web Graph (WAW)*, pp. 153–164.

Watts, D. J., and S. H. Strogatz. 1998. "Collective dynamics of 'small-world' networks". *Nature* vol. 393 (6684), pp. 440–442.

Xulvi-Brunet, R., and I. M. Sokolov. 2004. "Reshuffling scale-free networks: From random to assortative". *Physical Review E* vol. 70 (6), pp. 066102.

AUTHOR BIOGRAPHIES

HASANUZZAMAN BHUIYAN is a Ph.D. candidate in the Network Dynamics and Simulation Science Laboratory and Department of Computer Science at Virginia Tech. His email address is mhb@bi.vt.edu.

MALEQ KHAN is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science at Texas A&M University—Kingsville. His email address is maleq.khan@tamuk.edu.

MADHAV MARATHE is the Director of the Network Dynamics and Simulation Science Laboratory and Professor of Computer Science at Virginia Tech. His email address is mmarathe@bi.vt.edu.